Writing Tcl programs in the Medusa Applications Environment

Frank Stajano

Olivetti Research Limited
24a, Trumpington Street,
Cambridge CB2 1QA, England (U.K.)

Phone: (+ 44 223) 34.30.00
Fax: (+44 223) 31.35.42
E-mail: fstajano@cam-orl.co.uk

Abstract

Medusa is an applications environment for distributed multimedia which has been designed and developed at the Olivetti Research Laboratory in Cambridge, U.K. The software building blocks, or modules, are written in C++, while the applications that create networks of modules and make useful things with them are written in Tcl/Tk/Tcl-DP.

The Medusa system

The backbone of the Medusa system is an ATM network which, by virtue of its high capacity and low latency, is capable of shipping multiple simultaneous audio and video streams between the various peripherals and workstations. The multimedia devices such as cameras, microphones and speakers are connected directly to the network rather than being physically attached to a specific workstation. This provides uniform and unconstrained scalability to a multistream setup: to equip a workstation with multiple cameras, so that the viewer at the other end can select between a talking head and several wide-angle views of the office, all that is required is to plug more cameras into the network wiring. The network becomes the important data bus of the system, taking over the workstation's bus. And looking through a remote camera does not have any extra performance cost with respect to looking through a local camera.

A Medusa workstation equipped with multiple cameras, speakers and microphones. Note the ATM network switches in the lower right, just under the video bricks.

The main features of Medusa are modularity and the ability to process the raw data. The software architecture is such that every device in the system — source, sink or unit of processing — is represented by a software "module"; modules can be connected together and data will flow through them. The behaviour of the system can be dynamically changed by inserting or removing modules from these pipelines. From a software point of view, modules are self-contained entities that can be written independently of each other. A video compression module may be written entirely in software and then later replaced by another module that instead drives a hardware compression card. The system never uses any video overlay techniques: all the multimedia data actually goes through the modules; this enables us to include analysers and processors in the pipelines. One interesting example of this is an application where the user can position sound in a quadraphonic space by waving a hand in front of a camera whose output is piped into a "virtual mouse" image analysis module.

This laboratory has developed numerous "ATM Direct Peripherals" such as cameras, video tiles, audio AD/DA cards, and disk arrays; these devices are built as small independent networkable computers, all based on the same general purpose board which contains a network interface and an ARM processor running our in-house micro-kernel, ATMos. Modules, being the software components that can handle the raw multimedia data, generally live in close contact with the hardware they control; they are written in C++, sometimes with an inner assembler core for the most time-critical sections like handling audio samples. Every ATM Direct Peripheral runs some ATMos processes that contain Medusa modules and uniformly expose the functionality of the device to the rest of the Medusa system.

Medusa is also integrated with the Active Badge™ personnel and equipment location system at the ORL laboratory. One can ask the system to make a connection to a specific user; the system will then find the location of the user, find out what Medusa devices are currently available at that location, and make the appropriate connections to these devices. No manual database updates are required if equipment is moved from one office to another.

The integration of Medusa and Tcl

For obvious efficiency reasons we do not use Tcl to handle the multimedia data; instead we use it as a configuration language to control the way in which modules should be connected together. Modules are not written in Tcl, but whole applications, which use the modules as prepackaged components, are.

The Medusa modules have been made accessible to Tcl using the same object-oriented approach that Tk itself uses for its X widgets. Consider a standard Tk widget: its name is also a command that can be used to control the behaviour of the widget. The same thing happens with Medusa modules: after creation, the module name becomes a new Tcl command that accepts subcommands. These subcommands, or methods, allow the programmer to configure the module, connect it to another module, query its state and so on.

Every module has a set of attributes (internal variables) which can be read and written with the *getattributes* and *setattributes* methods. Changing an attribute may have the side effect of modifying the behaviour of a module,

as in the case of the *frame_rate* attribute of a camera module. The attributes can be changed by the module itself, and can indeed even be read-only, as in the case of the *level* attribute of a VU-meter module. Using the *watchattributes* method it is possible to register a Tcl callback that will be invoked whenever the value of an attribute changes, much in the same way as can be done with the Tcl command *trace* on ordinary variables. This allows a graphical Tk-based VU-meter to follow in real time the variations reported by the corresponding module without having to poll the module to periodically retrieve the value being monitored. Since modules can also model external devices like a remote control, the *watchattributes* mechanism also provides a way to activate Tcl procedures using input devices that are not tied or even related to the workstation. “Buttons”, “sliders” and other controllers are no longer limited to on-screen widgets that have to be clicked upon with the mouse: they can be real-world objects that combine flexible software-defined semantics with a physical, tactile feel.

To provide higher efficiency all the module methods support an asynchronous mode of operation, where the program requests an action and the command returns immediately, long before the action (which goes on on its own) has completed. This as opposed to the synchronous case where a command only returns after it has run to completion and can report whether it succeeded or failed. To make this asynchronous mechanism useful, every module method accepts the *-success* and *-failure* options, which specify Tcl callback commands to be evaluated when the main command has reached completion. For these commands where a return value is meaningful, the returned value is appended to the callback command before evaluation.

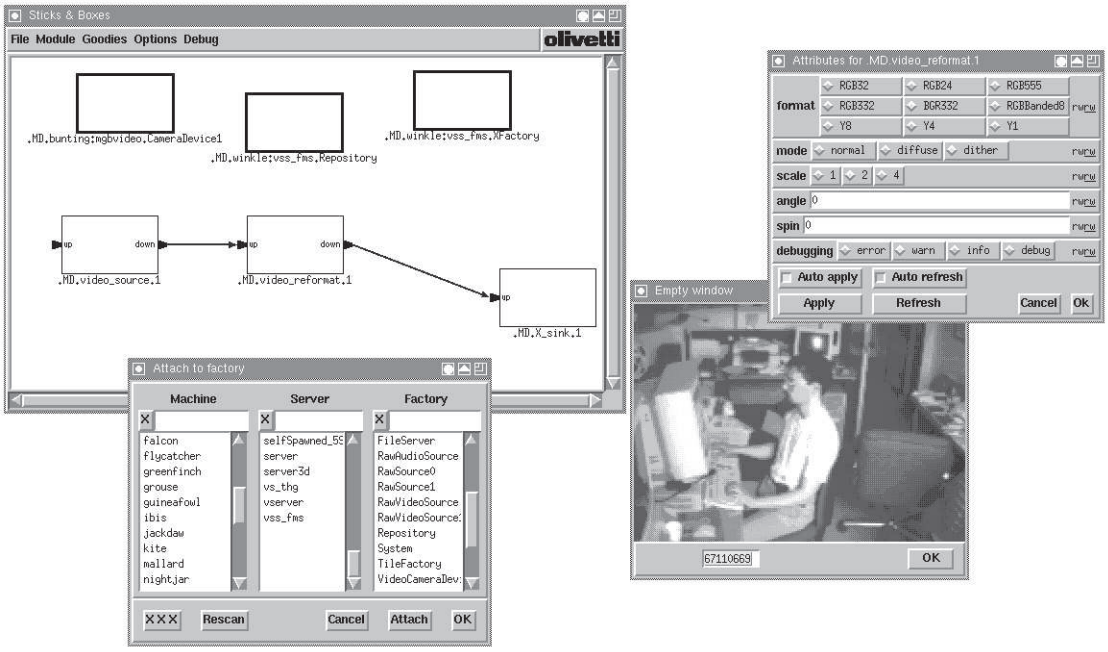
Medusa applications

The full power of Tcl and Tk is available to write complete multimedia applications using the modules as building blocks. So far two major applications have been written: *Sticks And Boxes*, a test tool that gives visual access to the modules, and *MDphone*, a multistream videophone system.

Sticks And Boxes

This application provides a graphical user interface to the task of creating modules and plugging them together. The name of the program refers to the visual appearance of the interface elements, which are implemented as Tk canvas items: modules are shown as boxes with nozzles as their data ports, while data connections are shown as sticks joining the boxes. A browser lets you examine all the module factories currently available anywhere on the network to find the ones that will create the desired modules. A few mouse clicks are sufficient to instantiate and connect a microphone module here and a speaker module next door, effectively creating a simple audio link between two offices.

Once a module has been created, clicking on it produces a panel exposing the attributes, which can then be inspected and modified interactively even while the multimedia data is flowing. *Sticks And Boxes* has no embedded knowledge about the various modules, but it will still correctly expose the attributes for every module that it comes across, presenting an interface that is appropriate to the type of the attribute being shown. Boolean attributes are displayed as checkbuttons, enumerations are displayed as a set of radio buttons, integers within a certain range (such as a volume control) are displayed as sliders and so on. For each mod-



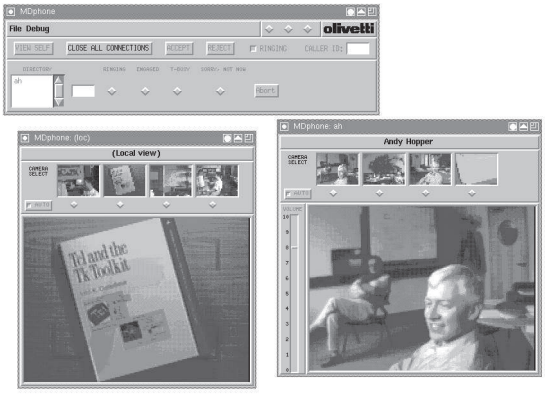
Sticks And Boxes *shows a pipeline of modules sending pictures from a network camera to an X11 window. The modules with the thick border are factories. The factory browser window and an attribute panel window are also visible.*

ule, *Sticks* builds a control panel tailored to the particular attributes of the module. This is achieved using a special read-only text attribute that every module has, which lists all the attributes of the module together with their type, default value, read/write status and so on: every module is thus self-describing and tools such as *Sticks* maintain their generality as new modules are written.

MDphone

The Medusa video phone system rejects the conventional constraint of using a single camera and microphone per multimedia workstation. Standard Medusa-equipped workstations feature an X11 display, four hi-fi directional microphones, four hi-fi speakers and four digital colour cameras. One of the cameras provides the “talking head” view when the user is sitting at the computer; another camera, vertically mounted on a wall stand, focuses on a free area of the desk where the user can display a document or a book; the other cameras give a wide-angle view of the whole room from different points of view. The same goes for the microphones which, by virtue of their positioning, can pick up high quality speech even when the user is not in front of the computer. This encourages a wide-band communication, where the user can walk across the room and draw up a diagram at the whiteboard while still talking to the correspondent at the other end.

The system has been implemented as a distributed application using Tcl-DP. Every phone is a separate process, running on its user’s computer. There is a central server, conceptually equivalent to the phone exchange, to which all the phones are connected. A list of the currently connected phones is broadcast by the server whenever there is a change. The *dp_atclose* callbacks are widely used to ensure that the sudden death of a process will not confuse the remaining running processes.



A two-way multistream conversation using MDphone

The user interface tries to present all those streams in an organic and meaningful arrangement, avoiding an excessive proliferation of controls. In view of the extension to a multi-way conference system, all the views from the same user have been grouped in a single toplevel window with many subframes. All the views are shown in small frames and one of them — selectable by the user — is replicated in a larger frame. There are plans to make this selection automatic if desired by using an image processing module that would recognise which camera the user is facing; this is already being done for the audio, where the system dynamically switches to the microphone with the loudest signal. Moving the composite window on the desktop actually pans the sound in the quadraphonic image, so that in a multi-way conference the audio positioning of the participants is coherent with the positioning of their respective windows.

Advantages and disadvantages of using Tcl in such a project

Using Tcl has been a great plus for this project, especially when compared to the original solution of adopting an ad-hoc configuration language to plumb modules together. With respect to that alternative, Tcl has given us a much greater flexibility and the expressive power of a full programming language.

The main drawback has been the difficulty of managing large programs in a programming environment that provides no static consistency checks. A consistent coding style and the use of self-contained autoloading libraries may help in keeping the problem under control, but explicit checking tools would be very welcome.

Some efficiency problems were experienced, but they were largely offset by the convenience with which a running prototype could be produced. Ease of prototyping has been a great asset in experimenting with new ideas and has proved to be the right approach for a research environment, so much that Tcl/Tk has rapidly spread to other unrelated projects in our lab.

Acknowledgements

I would like to thank Stuart Wray, who designed and implemented the Tcl interface to Medusa, and the many colleagues at Olivetti Research who provided helpful feedback on the usability and desirable functionalities of the Medusa applications I wrote.

Frank Stajano obtained his degree in electronic engineering in 1991 from the university of Rome, Italy. He joined Olivetti Research Limited in late 1992, where he has been working since as the Tcl applications programmer for the Medusa project. His interests include Walt Disney comics and human-computer interfaces.